Google Code

Archive

Search this site

Projects Search About

Project

Source

Issues

Wikis

Downloads

pixilang - PixilangV3Manual_RU.wiki

Export to GitHub

Что такое Pixilang

Pixilang - кросс-платформенный язык программирования для небольших графических и звуковых приложений. Примеры применения: арт-эксперименты, демки (demoscene), синтезаторы, игры. Концепция языка разработана в 2006 году Александром Золотовым (NightRadio) и Михаилом Разуваевым (Goglus).

Pixilang-программы хранятся в текстовых файлах (UTF8) с расширением .txt или .pixi. Поэтому вы можете использовать любой текстовый редактор для создания таких программ. Pixilang не имеет встроенного редактора. После старта появляется файловый диалог, в котором нужно указать, где лежит запускаемая pixi-программа.

Ключевые особенности: * простые правила, низкий порог вхождения; * поддержка графических и звуковых форматов файлов; * программу можно писать без объявления функций, просто списком инструкций с условными переходами; * сразу после старта программе выделяется чистый экран (или окно), к которому можно обращаться как к массиву пикселей, или использовать готовые граф. примитивы.

Запуск из командной строки

При запуске из командной строки Pixilang принимает дополнительные опции в приведенном ниже формате. pixilang [options] [source_file_name] [options] -c генерация байт-кода; будет создан файл source_file_name.pixicode. Примечание: файлы в формате *.pixicode привязаны к архитектуре и могут неправильно работать на других устройствах.

Основы

В основе Pixilang - контейнеры (или pixi-контейнеры, как их иногда называют) и переменные.

Что такое контейнер? Если в двух словах, то это двумерный массив, таблица из X колонок и Y строк. Каждая ячейка этой таблицы - число определенного формата. Формат задан один на весь контейнер. Например, ячейки могут хранить цвета пикселей, тогда контейнер превращается в картинку. Контейнер с таким же успехом может быть строкой текста, куском звука и т.д. Если вы знакомы с другими языками программирования, то считайте контейнер массивом, состоящим из (X*Y) ячеек. Каждый контейнер после создания имеет свой порядковый номер.

Структура контейнера: * двумерный массив (таблица) элементов контейнера; * key color - цвет, который будет отображаться, как прозрачный; * ссылка на контейнер (должен быть типа INT8) с альфа-каналом - для изображений с плавными изменениями прозрачности; * дополнительные данные: * свойства (используйте функции get_prop(), set_prop(), remove_props() или оператор . (точка) для доступа к ним); * анимация (используйте функции для анимации контейнеров).

Новый контейнер можно получить используя функцию new() или какую-то другую специальную функцию, возвращающую блок данных. Контейнер удаляется двумя способами: * пользователем при помощи функции remove(); * автоматически после завершения программы. В Pixilang отсутствует автоматический сборщик мусора, поэтому будьте внимательны - каждый новый контейнер отнимает память до тех пор, пока вы не удалите этот контейнер.

Переменная - имя ячейки памяти, в которой хранится одно знаковое целое 32-битное число (например, 25) или 32-битное число с плавающей запятой (например, 33.44). Локальные переменные (с символом $ перед именем) доступны только в рамках одной функции, в которой эти переменные определены. Глобальные переменные (без символа $) доступны в любом месте программы.

Числа можно описывать в различных форматах. Примеры: * 33 - обычное целое десятичное; * 33.55 - десятичное с плавающей запятой; * 0xA8BC - шестнадцатеричное; * 0b100101011 - двоичное; * #FF9080 - цвет, как в HTML; общий формат такой: #RRGGBB, где RR - яркость красного, GG - яркость зеленого; BB - яркость голубого. В каком бы формате вы не описали число, внутри Pixilang оно все равно будет 32-битным целым или 32-битным с плавающей запятой.

Простейшие примеры применения контейнеров и переменных: x = new(4) //Создаем контейнер из 4x пикселей. Сохраняем номер контейнера в переменной x. x[2] = WHITE //Присваиваем пикселю номер 2 белый цвет. remove(x) //Удаляем контейнер c = new(4, 4) //Создаем 2D контейнер 4 на 4 пикселя. Сохраняем номер контейнера в переменной c. c[2, 2] = WHITE //Присваиваем пикселю с координатами 2,2 белый цвет. remove(c) //Удаляем контейнер str = "Hello" //"Hello" - это строковый контейнер, состоящий из пяти 8-битных символов (кодировка UTF-8). //Подобные контейнеры-строки создаются автоматически на этапе компиляции программы. //Удалять их вручную так, как это сделано в предыдущем примере, не надо. //Контейнеру со строкой "Hello" автоматически присвоится порядковый номер. //Например, это будет номер 4. //Тогда код str = "Hello" будет равноценен коду str = 4. str[0] = 'h' //Меняем самую первую букву в строке. Было - 'H'. Станет - 'h'. a = 4 //Глобальная переменная fn function() { $k = 2 //Локальная переменная function2 = { //Определяем еще одну функцию $x = 899.334 //Локальная переменная //В этом месте $k недоступна, т.к. находится в другой функции } //В этом месте $x недоступна } //В этом месте $k и $x недоступны

Имена файлов и директорий

Ниже приведены примеры, показывающие основные правила оформления имен файлов и директорий. ``` //Файл спрятан в нескольких директориях относительно текущего местоположения пикси-программы: "folder1/folder2/folder3/prog.pixi"

//Файл лежит в текущей рабочей папке Pixilang // для iOS: documents; // для WinCE: корень файловой системы (/); // для остальных систем: в той же самой папке, в которой лежит pixilang или pixilang.exe; "1:/prog.pixi"

//Файл лежит в директории пользователя (например, в Linux это будет /home/alex): "2:/prog.pixi"

//Файл лежит во временной директории: "3:/prog.pixi" ```

Встроенные операторы

Рассмотрим операторы на конкретных примерах.

``` //Условные операторы if, else if a == b { /**Код в этом месте выполняется, если a равно b**/ } else { /**Код в этом месте выполняется в противном случае (a не равно b)**/ } if x == 4 && y == 2 { /**Код в этом месте выполняется, если x равно 4 и y равно 2**/ }

//Оператор цикла: while a = 0 while( a < 3 ) { //Код в этом месте выполняется, если a меньше 3 a + 3 }

//Операторы цикла: while, break a = 0 while( a < 100 ) { //Код в этом месте выполняется, если a меньше 100 if a == 10 { break } //Если a = 10, то разрываем цикл оператором break //Для остановки нескольких вложенных циклов сразу можно использовать оператор breakX, //где X - глубина. Например break2 остановит два цикла. //А при помощи оператора breakall можно остановить все циклы, //которые активны в данный момент для текущего потока выполнения. a + 1 }

//Операторы цикла: while, continue a = 0 b = 0 while( a < 100 ) { //Код в этом месте выполняется, если a меньше 100 if a == 10 { a + 1 continue } //Если a = 10, то переходим к следующей итерации цикла // (игнорируем следующие две строчки кода) a + 1 b + 1 }

//Операторы перехода: go, goto m1: a + 1 goto m1 //Переход на метку m1

//Операторы остановки: halt, stop halt //В этом месте программа останавливается

//Оператор подключения: include include "prog2.txt" //В этом месте подключаем код из файла prog2.txt

//Оператор определения функции: fn fn fff( $x, $y ) //Определяем функцию fff с параметрами $x и $y { //Код функции fff ret //Простой выход из функции ret( 4 ) //Выход из функции с возвращением значения 4 } ```

Ниже приведена таблица математических операторов. Приоритет 0 - наивысший, такие операции будут выполняться в первую очередь. | Приоритет | Оператор | Описание | Результат | Пример | |:———|:———|:———|:———|:——| | 0 | % | Деление по модулю | Целое число | a = b % 4 | | 0 | / | Деление | Число с плавающей запятой | a = b / 4 | | 0 | div | Целочисленное деление | Целое число | a = b div 4 | | 0 | * | Умножение | Зависит от операндов | a = b * 4 | | 1 | + | Сложение | Зависит от операндов | a = b + 4 | | 1 | - | Вычитание | Зависит от операндов | a = b - 4 | | 2 | >> | Битовый сдвиг вправо | Целое число | a = b >> 4 | | 2 | << | Битовый сдвиг влево | Целое число | a = b << 4 | | 3 | == | Равно | Целое число 1 или 0 | if a == b {} | | 3 | != | Не равно | Целое число 1 или 0 | if a != b {} | | 3 | < | Меньше | Целое число 1 или 0 | if a < b {} | | 3 | > | Больше | Целое число 1 или 0 | if a > b {} | | 3 | <= | Меньше или равно | Целое число 1 или 0 | if a <= b {} | | 3 | >= | Больше или равно | Целое число 1 или 0 | if a >= b {} | | 4 | \| | Побитовая операция ИЛИ (OR) | Целое число | a = b \| 4 | | 4 | ^ | Побитовая операция исключающего ИЛИ (XOR) | Целое число | a = b ^ 4 | | 4 | & | Побитовая операция И (AND) | Целое число | a = b & 4 | | 5 | \|\`\| | Логическая операция ИЛИ (OR) | Целое число 1 или 0 | if a \|\`\| b {} | | 5 | && | Логическая операция И (AND) | Целое число 1 или 0 | if a && b {} |

## Встроенные константы

### Типы контейнеров

Контейнер может содержать элементы одного из нижеперечисленных типов.

64-битные типы в текущей версии не поддерживается, но поддержка может быть включена при самостоятельной сборке Pixilang из исходников. Для включения необходимо внести небольшие правки в файле pixilang.h в разделе Configuration.
```

Only 2 pages have been converted.
Please go to https://docs.zone and Sign Up to convert all pages.